Informe Técnico

Nombre: Gestión de campeonatos de béisbol

Autores:
Ariadna Velázquez Rey C311
Lía López Rosales C312
Carlos Daniel Largacha Leal C312
Gabriel Andrés Pla Lasa C311
Raidel Miguel Cabellud Lizaso C311

Diccionario de datos

Tabla: User

Campo	Tipo de Dato	Descripción	Restricciones
id	AutoField (PK)	Identificador único del usuario.	Primary Key
email	EmailField	Correo electrónico del usuario.	Único, máximo 255 caracteres.
rol_id	ForeignKey (Rol)	Rol asignado al usuario.	No nulo, relación con la tabla `Rol`.
TD_id	ForeignKey (TechnicalDirector)	Director técnico asociado al usuario.	Nulo permitido, relación con `TechnicalDirector`.
password	CharField	Contraseña del usuario.	Máximo 128 caracteres.

Restricciones adicionales:

- El campo `TD_id` no puede ser nulo si el `rol_id` es "Director Técnico".

—

Tabla: Rol

Campo	Tipo de Dato	Descripción	Restricciones
id	AutoField (PK)	Identificador único del rol.	Primary Key
type	CharField	Tipo de rol (Director Técnico, Admin, Usuario).	Máximo 50 caracteres, opciones predefinidas.

—

Tabla: TechnicalDirector

Campo	Tipo de Dato	Descripción	Restricciones
id	AutoField (PK)	Identificador único del director técnico.	Primary Key
direction_team	ForeignKey (DirectionTeam)	Equipo de dirección asociado.	Nulo permitido, relación con `DirectionTeam`.
W_id	ForeignKey (Worker)	Trabajador asociado al director técnico.	No nulo, relación con `Worker`.

Restricciones adicionales:

- Clave primaria compuesta: `W_id` e `id`.

—

Tabla: Worker

Campo	Tipo de Dato	Descripción	Restricciones
id	AutoField (PK)	Identificador único del trabajador.	Primary Key
P_id	ForeignKey (Person)	Persona asociada al trabajador.	No nulo, relación con `Person`.
DT_id	ForeignKey (DirectionTeam)	Equipo de dirección asociado.	Nulo permitido, relación con `DirectionTeam`.

Restricciones adicionales:

- Clave primaria compuesta: `P_id` e `id`.

—

Tabla: DirectionTeam

Campo	Tipo de Dato	Descripción	Restricciones
id	AutoField (PK)	Identificador único del equipo de dirección.	Primary Key
Team_id	ForeignKey (Team)	Equipo asociado al equipo de dirección.	No nulo, relación con `Team`.

—

Tabla: Team

Campo	Tipo de Dato	Descripción	Restricciones
id	AutoField (PK)	Identificador único del equipo.	Primary Key
name	CharField	Nombre del equipo.	Máximo 100 caracteres.
color	CharField	Color del equipo.	Máximo 50 caracteres.
initials	CharField	Iniciales del equipo.	Máximo 10 caracteres.
representative_entity	CharField	Entidad representativa del equipo.	Máximo 100 caracteres.

Tabla: Person

Campo	Tipo de Dato	Descripción	Restricciones
id	AutoField (PK)	Identificador único de la persona.	Primary Key
CI	IntegerField	Cédula de identidad de la persona.	Único.
age	IntegerField	Edad de la persona.	
name	CharField	Nombre de la persona.	Máximo 100 caracteres.
lastname	CharField	Apellido de la persona.	Máximo 100 caracteres.

Tabla: Position

Campo	Tipo de Dato	Descripción	Restricciones
id	AutoField (PK)	Identificador único de la posición.	Primary Key
name	CharField	Nombre de la posición.	Máximo 100 caracteres.

Tabla: BaseballPlayer

Campo	Tipo de Dato	Descripción	Restricciones
id	AutoField (PK)	Identificador único del pelotero.	Primary Key
P_id	ForeignKey (Person)	Persona asociada al pelotero.	No nulo, relación con `Person`.
batting_average	FloatField	Promedio de bateo del pelotero.	
years_of_experience	IntegerField	Años de experiencia del pelotero.	
pitcher	ForeignKey (Pitcher)	Lanzador asociado al pelotero.	Nulo permitido, relación con `Pitcher`.

Tabla: Pitcher

Campo	Tipo de Dato	Descripción	Restricciones
P_id	ForeignKey (BaseballPlayer)	Pelotero asociado al lanzador.	No nulo, relación con `BaseballPlayer`.
dominant_hand	CharField	Mano dominante del lanzador.	Opciones: 'izquierda', 'derecha', 'ambas'.
No_games_won	PositiveIntegerField	Número de juegos ganados.	
No_games_lost	PositiveIntegerField	Número de juegos perdidos.	
running_average	PositiveIntegerField	Promedio de carreras permitidas.	

Tabla: BPParticipation

Campo	Tipo de Dato	Descripción	Restricciones
BP_id	ForeignKey (BaseballPlayer)	Pelotero que participa en la serie.	No nulo, relación con `BaseballPlayer`.
series	ForeignKey (Series)	Serie en la que participa el pelotero.	No nulo, relación con `Series`.
team_id	ForeignKey (Team)	Equipo al que pertenece el pelotero.	No nulo, relación con `Team`.

Restricciones adicionales:

- Clave primaria compuesta: `BP_id` y `series`.

—

Tabla: LineUp

Campo	Tipo de Dato	Descripción	Restricciones
id	AutoField (PK)	Identificador único de la alineación.	Primary Key
team_id	ForeignKey (Team)	Equipo asociado a la alineación.	No nulo, relación con `Team`.

Restricciones adicionales:

- Clave primaria compuesta: `id` y `team_id`.

—

Tabla: PlayerInLineUp

Campo	Tipo de Dato	Descripción	Restricciones
line_up	ForeignKey (LineUp)	Alineación a la que pertenece el jugador.	No nulo, relación con `LineUp`.
player_in_position	ForeignKey (PlayerInPosition)	Jugador en una posición específica.	No nulo, relación con `PlayerInPosition`.

Restricciones adicionales:

- Clave primaria compuesta: `line_up` y `player_in_position`.

—

Tabla: PlayerInPosition

Campo	Tipo de Dato	Descripción	Restricciones
BP_id	ForeignKey (BaseballPlayer)	Pelotero en una posición.	No nulo, relación con `BaseballPlayer`.
position	ForeignKey (Position)	Posición del pelotero.	No nulo, relación con `Position`.
effectiveness	FloatField	Efectividad del pelotero en la posición.	

Restricciones adicionales:

- Clave primaria compuesta: `BP_id` y `position`.

—

Tabla: Season

Campo	Tipo de Dato	Descripción	Restricciones
id	AutoField (PK)	Identificador único de la temporada.	Primary Key
name	CharField	Nombre de la temporada.	

—

Tabla: Series

Campo	Tipo de Dato	Descripción	Restricciones
id	AutoField (PK)	Identificador único de la serie.	Primary Key
season	ForeignKey (Season)	Temporada a la que pertenece la serie.	No nulo, relación con `Season`.
name	CharField	Nombre de la serie.	Máximo 100 caracteres.
type	CharField	Tipo de serie.	Máximo 50 caracteres.
init_date	DateTimeField	Fecha de inicio de la serie.	
end_date	DateTimeField	Fecha de finalización de la serie.	

Restricciones adicionales:

- Clave primaria compuesta: `season` e `id`.
- `init_date` debe ser anterior a `end_date`.

—

Tabla: Game

Campo	Tipo de Dato	Descripción	Restricciones
id	AutoField (PK)	Identificador único del juego.	Primary Key
local	ForeignKey (TeamOnTheField)	Equipo local.	No nulo, relación con `TeamOnTheField`.
date	DateTimeField	Fecha del juego.	
rival	ForeignKey (TeamOnTheField)	Equipo rival.	No nulo, relación con `TeamOnTheField`.
series	ForeignKey (Series)	Serie a la que pertenece el juego.	No nulo, relación con `Series`.
score	ForeignKey (Score)	Puntuación del juego.	Nulo permitido, relación con `Score`.

Restricciones adicionales:

- Clave primaria compuesta: `local` y `date`.
- `local` no puede ser igual a `rival`.

—

Tabla: Score

Campo	Tipo de Dato	Descripción	Restricciones
id	AutoField (PK)	Identificador único del marcador.	Primary Key
winner	ForeignKey (Team)	Equipo ganador.	No nulo, relación con `Team`.
loser	ForeignKey (Team)	Equipo perdedor.	No nulo, relación con `Team`.
w_points	PositiveIntegerField	Puntos del equipo ganador.	
l_points	PositiveIntegerField	Puntos del equipo perdedor.	

Restricciones adicionales:

- `winner` no puede ser igual a `loser`.
- `w_points` debe ser mayor o igual a `l_points`.

—

Tabla: PlayerSwap

Campo	Tipo de Dato	Descripción	Restricciones
id	AutoField (PK)	Identificador único del cambio.	Primary Key
old_player	ForeignKey (BaseballPlayer)	Jugador reemplazado.	No nulo, relación con `BaseballPlayer`.
date	DateTimeField	Fecha del cambio.	
new_player	ForeignKey (BaseballPlayer)	Jugador que entra.	No nulo, relación con `BaseballPlayer`.
position	ForeignKey (Position)	Posición del cambio.	No nulo, relación con `Position`.
game_team	ForeignKey (TeamOnTheField)	Equipo en el campo.	No nulo, relación con `TeamOnTheField`.

Restricciones adicionales:

- Clave primaria compuesta: `old_player` y `date`.

—

Tabla: StarPlayer

Campo	Tipo de Dato	Descripción	Restricciones
id	AutoField (PK)	Identificador único del jugador estrella.	Primary Key
series	ForeignKey (Series)	Serie en la que es jugador estrella.	No nulo, relación con `Series`.
position	ForeignKey (Position)	Posición del jugador estrella.	No nulo, relación con `Position`.
BP_id	ForeignKey (BaseballPlayer)	Pelotero estrella.	No nulo, relación con `BaseballPlayer`.

Restricciones adicionales:

- Clave primaria compuesta: `series` y `position`.

Tabla: TeamOnTheField

Campo	Tipo de Dato	Descripción	Restricciones
id	AutoField (PK)	Identificador único del equipo en el campo.	Primary Key
lineup_id	ForeignKey (LineUp)	Alineación asociada al equipo en el campo.	No nulo, relación con `LineUp`.

Esquema de las clases definidas

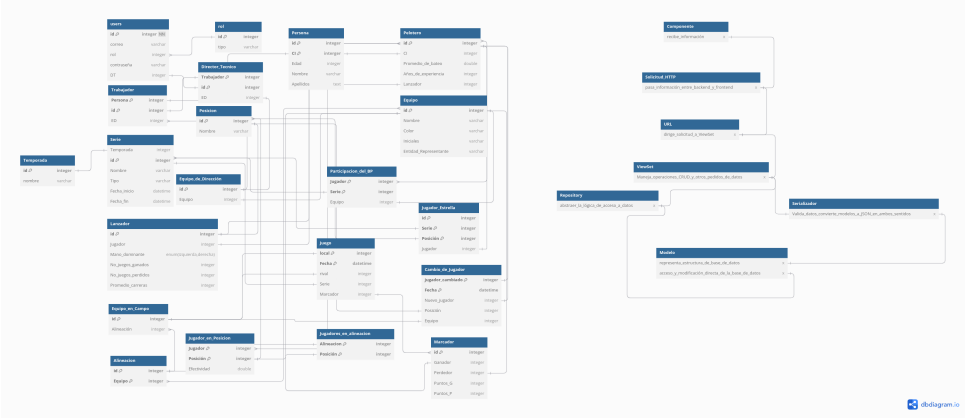

Figure 1: Diagrama de gestión de campeonatos de béisbol

La aplicación sigue un flujo cliente-servidor, donde el frontend (React) y el backend (Django Rest Framework) se comunican a través de solicitudes HTTP. A continuación, se describe el flujo de información y procesamiento en cada capa de la aplicación:

Frontend (React)

El frontend es responsable de la interfaz de usuario (UI) y la interacción con el usuario. Está construido con React, una biblioteca de JavaScript que permite crear interfaces dinámicas y reactivas. El flujo en el frontend es el siguiente:

Interfaz de Usuario (UI)

- React renderiza componentes que representan la interfaz gráfica de la aplicación.
- Los componentes pueden ser formularios, listas, tablas, botones, etc., que permiten al usuario interactuar con la aplicación.

Gestión del Estado

- React utiliza un sistema de estado (por ejemplo, con `useState`, `useReducer` o bibliotecas como Redux) para manejar los datos que se muestran en la interfaz.
- El estado se actualiza dinámicamente en respuesta a las acciones del usuario (por ejemplo, hacer clic en un botón o enviar un formulario).

Comunicación con el Backend

- Cuando el usuario realiza una acción que requiere datos del servidor (por ejemplo, cargar una lista de equipos o enviar un formulario), React realiza una solicitud HTTP (GET, POST, PUT, DELETE) al backend a través de la API REST.
- Para realizar estas solicitudes, se utiliza `fetch` o bibliotecas como Axios.

Renderizado Dinámico

- Una vez que el backend responde con los datos solicitados, React actualiza el estado de la aplicación y re-renderiza los componentes para reflejar los cambios en la interfaz.

Backend (Django Rest Framework)

El backend es responsable de la lógica de negocio, el procesamiento de datos y la comunicación con la base de datos. Está construido con Django Rest Framework (DRF), que proporciona herramientas para crear APIs RESTful de manera eficiente. El flujo en el backend es el siguiente:

Solicitud HTTP

- Cuando el frontend realiza una solicitud HTTP (por ejemplo, una petición GET para obtener datos o POST para enviar datos), Django recibe la solicitud y la procesa.

Enrutamiento (URLs)

- Django utiliza un sistema de enrutamiento (definido en el archivo `urls.py`) para dirigir la solicitud al ViewSet o Vista correspondiente.

Vistas (Viewsets)

- Los ViewSets (o Vistas) son responsables de manejar la lógica de la solicitud. En DRF, los ViewSets permiten realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) de manera simplificada.
- Las vistas interactúan con los serializadores para validar y transformar los datos entrantes o salientes.

Serializadores

Los serializadores en DRF son responsables de:

- Validar los datos enviados por el frontend (por ejemplo, datos de un formulario).
- Convertir los objetos de Django (modelos) en formatos como JSON para enviarlos al frontend.
- Convertir los datos JSON recibidos del frontend en objetos de Django para su procesamiento.

Modelos y Base de Datos

- Los modelos de Django representan la estructura de la base de datos. Cada modelo define una tabla y sus campos.
- Cuando es necesario acceder o modificar datos, las vistas interactúan con los modelos a través del ORM (Object-Relational Mapping) de Django.
- El ORM permite realizar consultas a la base de datos sin escribir SQL directamente, lo que simplifica el manejo de datos.

Repositorios

- Se utilizan repositorios para abstraer la lógica de acceso a datos. Los repositorios actúan como una capa intermedia entre las vistas y los modelos, centralizando las operaciones de la base de datos.

Respuesta HTTP

- Una vez que la vista procesa la solicitud y obtiene los datos necesarios, devuelve una respuesta HTTP al frontend. Esta respuesta suele ser un objeto JSON que contiene los datos solicitados o un mensaje de confirmación.

Flujo Completo de una Solicitud

A continuación, se describe el flujo completo de una solicitud típica en la aplicación:

1. **Usuario realiza una acción en el frontend:**
   - Por ejemplo, el usuario hace clic en un botón para cargar una lista de equipos.
2. **React realiza una solicitud HTTP:**
   - React utiliza Axios o `fetch` para enviar una solicitud GET al endpoint correspondiente en el backend (por ejemplo, `/api/teams/`).
3. **Django recibe la solicitud:**
   - Django dirige la solicitud al ViewSet correspondiente a través del archivo `urls.py`.
4. **ViewSet procesa la solicitud:**
   - El ViewSet utiliza el serializador para obtener los datos de la base de datos a través del modelo.
   - El serializador convierte los objetos de Django en JSON.
5. **Respuesta al frontend:**
   - El ViewSet devuelve una respuesta HTTP con los datos en formato JSON.
6. **React actualiza el estado y la interfaz:**
   - React recibe la respuesta, actualiza el estado de la aplicación y re-renderiza los componentes para mostrar los datos al usuario.

Esquema con el diseño de la aplicación

El diseño de la aplicación de Gestión de Campeonatos de Béisbol se fundamenta en la interacción entre los distintos actores y los casos de uso definidos a continuación.

Los actores principales de la aplicación son:

- **Administrador:** Responsable de gestionar los datos de series, equipos y jugadores, además de configurar el sistema y generar reportes.

- **Director Técnico:** Encargado de gestionar las alineaciones de los equipos y consultar estadísticas específicas.
- **Periodista/Usuario General:** Consulta las estadísticas y genera reportes basados en los datos almacenados en el sistema.

Y entre los principales casos de usos se encuentran:

- **Registrar datos en el sistema**
  - **Actor:** Administrador
  - **Descripción:** El administrador registra datos de temporadas, series, equipos y jugadores mediante formularios en la aplicación web.
- **Gestionar alineaciones de equipos**
  - **Actores:** Director Técnico y Administrador
  - **Descripción:** Permite al director técnico modificar la posición de los jugadores en la alineación inicial y realizar ajustes durante una serie.
- **Consultar estadísticas de jugadores y equipos**
  - **Actores:** Administrador, Director Técnico, Periodista/Usuario General
  - **Descripción:** El sistema permite acceder a estadísticas detalladas, como promedios de bateo, efectividad de los lanzadores y resultados de los equipos.
- **Obtener reportes personalizados**
  - **Actores:** Administrador, Director Técnico, Periodista/Usuario General
  - **Descripción:** Genera reportes que resumen el desempeño de jugadores y equipos en una serie.
- **Visualizar datos tabulares y gráficos**
  - **Actores:** Administrador, Director Técnico, Periodista/Usuario General
  - **Descripción:** Presentación visual de estadísticas mediante tablas y gráficos para un análisis claro y rápido.
- **Listar equipos por clasificación**
  - **Actores:** Administrador, Director Técnico, Periodista/Usuario General
  - **Descripción:** Permite obtener la clasificación de los equipos según los resultados de cada serie.
- **Modificar roles de usuarios**
  - **Actores:** Administrador
  - **Descripción:** El administrador puede asignar o modificar roles para los usuarios del sistema.
- **Exportar reportes a formato PDF**
  - **Actores:** Administrador, Director Técnico, Periodista/Usuario General
  - **Descripción:** Funcionalidad para guardar reportes generados en formato PDF, con soporte para otros formatos adicionales.

Solución propuesta

El presente sistema ha sido diseñado para optimizar la gestión de campeonatos de béisbol, facilitando el registro y administración de datos clave como equipos, jugadores, alineaciones y estadísticas de desempeño. Con el objetivo de mejorar la eficiencia en la organización y análisis de la información, la solución implementa una aplicación web con una interfaz intuitiva y herramientas avanzadas de procesamiento de datos.

La propuesta se fundamenta en una arquitectura N-Capas combinada con un Microkernel, permitiendo escalabilidad y modularidad en la funcionalidad del sistema. A través de un enfoque estructurado, se garantiza la seguridad y consistencia de la información, ofreciendo a cada actor del sistema (administradores, directores técnicos y periodistas) las herramientas necesarias para interactuar con los datos de manera eficiente.

Las capas propuestas son:

1. Capa de Presentación: Interfaz de Usuario

   La capa de presentación es el punto de contacto entre el sistema y los usuarios. En este caso, incluiría formularios, vistas y *dashboards* personalizados según los roles del usuario.

   - **Responsabilidades:**
     - Diseñar formularios dinámicos para el ingreso de datos por parte del administrador.
     - Ofrecer vistas diferenciadas para administradores, directores técnicos y periodistas.
     - Implementar filtros interactivos para que los usuarios consulten estadísticas y generen reportes personalizados.
     - Mostrar datos tabulares, como resultados históricos y estadísticas en tiempo real.

2. Capa de Negocio: Lógica del Sistema

   La capa de negocio maneja toda la lógica que regula el funcionamiento del sistema, separando las reglas de negocio de las funcionalidades específicas de la interfaz o los datos.

   - **Responsabilidades:**
     - Cálculo de estadísticas avanzadas como promedios de bateo o efectividad de lanzadores.
     - Validación de alineaciones en tiempo real para garantizar la coherencia de los equipos.
     - Gestión de roles y permisos, asegurando que cada usuario acceda únicamente a sus funcionalidades específicas.
     - Coordinación de las solicitudes de reportes y actualización de datos en la base de datos.
   - **Importancia:** Esta capa es el núcleo funcional del sistema, asegurando que las operaciones se realicen de manera eficiente y coherente.

3. Capa de Acceso a Datos: Persistencia

   La capa de acceso a datos es responsable de interactuar directamente con la base de datos. Esto incluye la consulta, modificación y almacenamiento de datos relevantes para el proyecto.

   - **Responsabilidades:**
     - Almacenar datos históricos y en tiempo real, como estadísticas de jugadores, resultados de partidos y alineaciones.
     - Proveer acceso a consultas complejas, como listar los equipos ganadores por temporada o generar reportes sobre las series con mayor cantidad de juegos.
     - Optimizar las operaciones de base de datos para soportar consultas frecuentes y minimizar tiempos de respuesta.

Y a su vez la implementación de Microkernel se encargaría:

- **Generación de Reportes:**

  - Los *plugins* generarían reportes en diferentes formatos, comenzando por PDF y con capacidad para añadir formatos como Excel o CVC.

- **Carga Dinámica de Plugins:**
  - Los *plugins* se detectan en tiempo de ejecución, permitiendo al sistema integrarlos sin interrupciones.

- **Integración con N-Capas:**
  - La capa de negocio gestiona las solicitudes de reportes, mientras que el núcleo del Microkernel se comunica con los *plugins* para entregar los resultados.

Además, se han seleccionado patrones específicos para mejorar aspectos clave como la gestión de datos, la comunicación entre capas y la presentación de información. Estos patrones proporcionan una estructura flexible que facilita la evolución del sistema y permite futuras mejoras sin comprometer su estabilidad.

Entre los patrones implementados se encuentran:

- **Repository Pattern** el cual que actúa como una capa intermedia entre la lógica de negocio y el acceso a datos. Entre sus principales caracteristicas se ecuentran:
  1. **Abstracción del acceso a datos:**
     - La capa de repositorio encapsula la lógica de consulta, lo que permite desacoplar los modelos y el acceso a la base de datos de la lógica de negocio.
     - Esto facilita realizar cambios en la estructura de la base de datos o en el mecanismo de almacenamiento sin afectar el resto del sistema.
  2. **Interfaz unificada:**
     - Proporciona una API consistente para las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) y consultas personalizadas.
  3. **Centralización de la lógica de acceso a datos:**
     - Todo el código relacionado con la interacción con la base de datos está centralizado, haciendo que sea más mantenible.
  4. **Facilidad de pruebas:**
     - Como la lógica de acceso a datos está encapsulada, es posible usar simulaciones (mocks) en las pruebas unitarias para probar la lógica de negocio sin depender de la base de datos real.

- **MVC Pattern** el cual se encuentra distribuido entre un cliente desarrollado en React y un servidor desarrollado en Django, siguiendo un enfoque RESTful. Esto implica una separación clara entre las responsabilidades del frontend y del backend:

  En el backend desarrollado con Django, los elementos del patrón MVC se distribuyen de la siguiente manera:

  - **Model:** Define los modelos de datos y la lógica de negocio asociada. Cada clase de modelo representa una tabla en la base de datos y permite realizar operaciones CRUD mediante el ORM.
  - **View:** En una aplicación RESTful, las vistas de Django (a través de Django REST Framework) funcionan como una capa de presentación de datos en formato JSON, no como vistas tradicionales en MVC que generan HTML.
  - **Controller:** En el contexto de Django, el rol de "Controlador" en el patrón MVC es desempeñado por las vistas (`views.py`). Estas se encargan de:
    * *Recibir solicitudes HTTP:* Las vistas manejan solicitudes del cliente como `GET`, `POST`, `PUT` o `DELETE`.
    * *Interactuar con el Modelo:* Realizan operaciones sobre los datos, como consultas o actualizaciones, utilizando los modelos definidos en el ORM de Django.
    * *Generar una Respuesta:* Transforman los datos en una respuesta adecuada (por ejemplo, JSON) que es enviada al cliente.

En el frontend desarrollado con React, los elementos del patrón MVC se adaptan de la siguiente manera:

- **View:** Renderiza la interfaz de usuario (UI) en el navegador del usuario, basándose en los datos obtenidos del backend. Los componentes de React representan esta capa de la arquitectura.
- **Controller:** En React, no hay un controlador explícito como en el MVC clásico, pero ciertas partes de una aplicación React pueden desempeñar funciones similares a un controlador como los componentes. Los componentes pueden manejar la lógica de presentación y también gestionar interacciones del usuario, como clics, envíos de formularios o cambios en el estado. Además, los componentes suelen enviar solicitudes a la API (backend) para obtener datos.

Otras informaciones relevantes

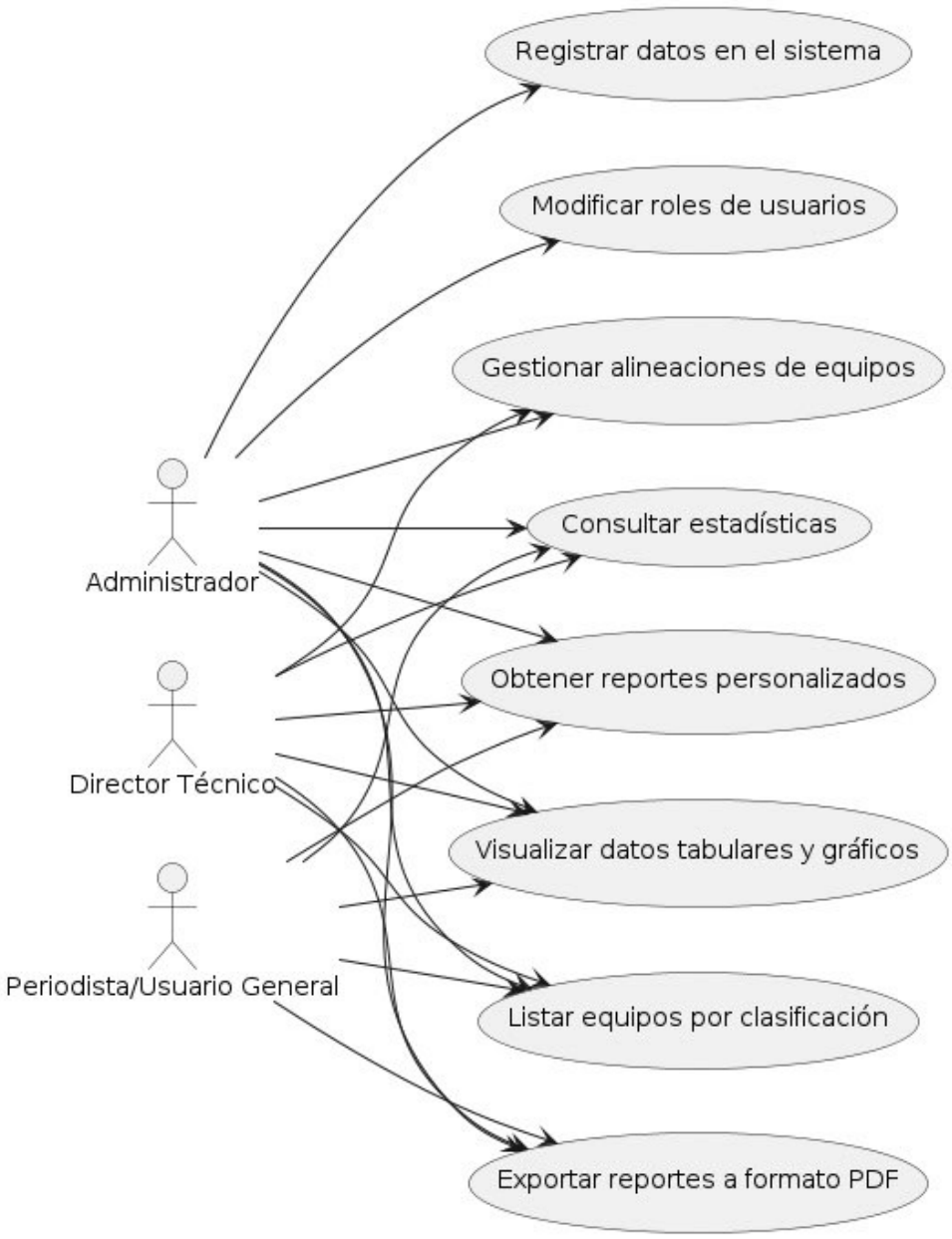

Figure 2: Diagrama de casos de uso